Project Assignment

& Solution

Cybersecurity Professional Program

Introduction to Python

for Security

Final Project

PY-07-LS1

ARP Spoof Detector

Note: Solutions for the instructor are shown inside the green box.

Project Objective

Implement learned skills to create an automated program that can be used in real-life scenarios.

Project Mission

Create a program that can detect active ARP Spoofing attacks on host machines.

Project Duration

2–4 hours

Requirements

- Working knowledge of Python
- Working knowledge of Python’s OS module

Resources

- Environment & Tools
    - Windows, macOS, and Linux
        - PyCharm
        - Python 3
    - Kali Linux ISO

Project Scenario

In the past few days, the HackRS company has been experiencing network connection issues, such as low bandwidth and disconnections. The IT team couldn't find the cause of the issue, and the company decided to ask for your assistance.

You suspect the network is under an On-Path attack. As a Python script expert, you need to create a script that automatically identifies ARP Spoofing behavior on workstations.

In the first stage, the script should read a station's ARP table and extract the address from it. The script should search for MAC duplications in table entries and log every ARP Spoofing event in the next stage.

Project Task 1: ARP Table Extraction

The first part of the program must handle the extraction of the ARP table data. The data must be saved in a dynamic structure in which the IP address corresponds to the MAC address. The structure is required for later comparison operations to identify duplicated addresses.

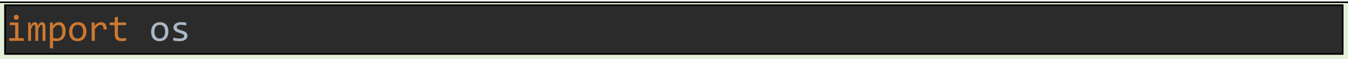
1 Plan a function that will extract the ARP table data from the machine. How can Python access this type of data? How should the data be saved for later use?

Since the ARP table is an operating system component, the OS module must be imported to enable ARP table data printing.

The ARP table data should be saved in a variable and then divided into list form. The list structure must iterate over the table's rows and treat each one individually.

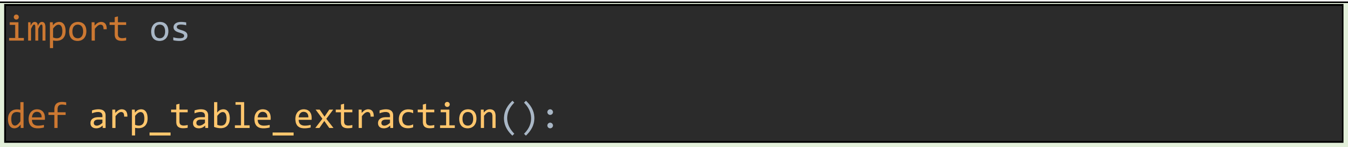
The list should be filtered from unnecessary data and saved in a dictionary that lists IP addresses corresponding to MAC addresses throughout iteration over ARP data lines.

2 Import the required modules for the program.



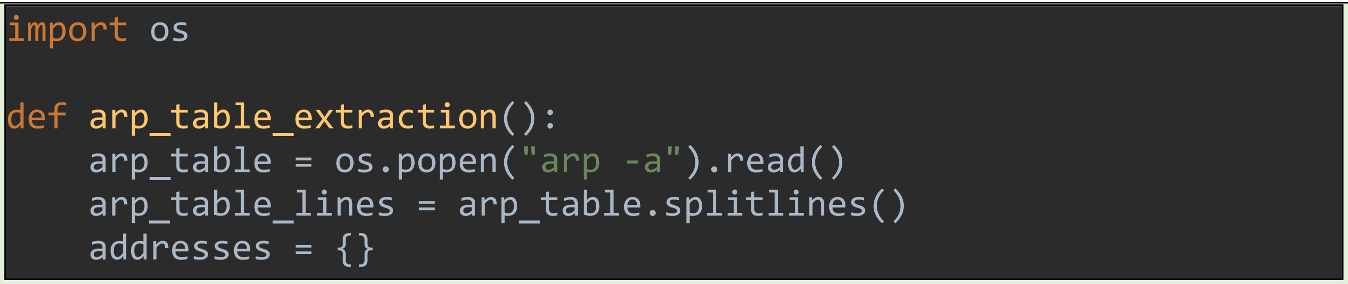
```python
import os
```

3 Define a function that will handle the ARP table data extraction.



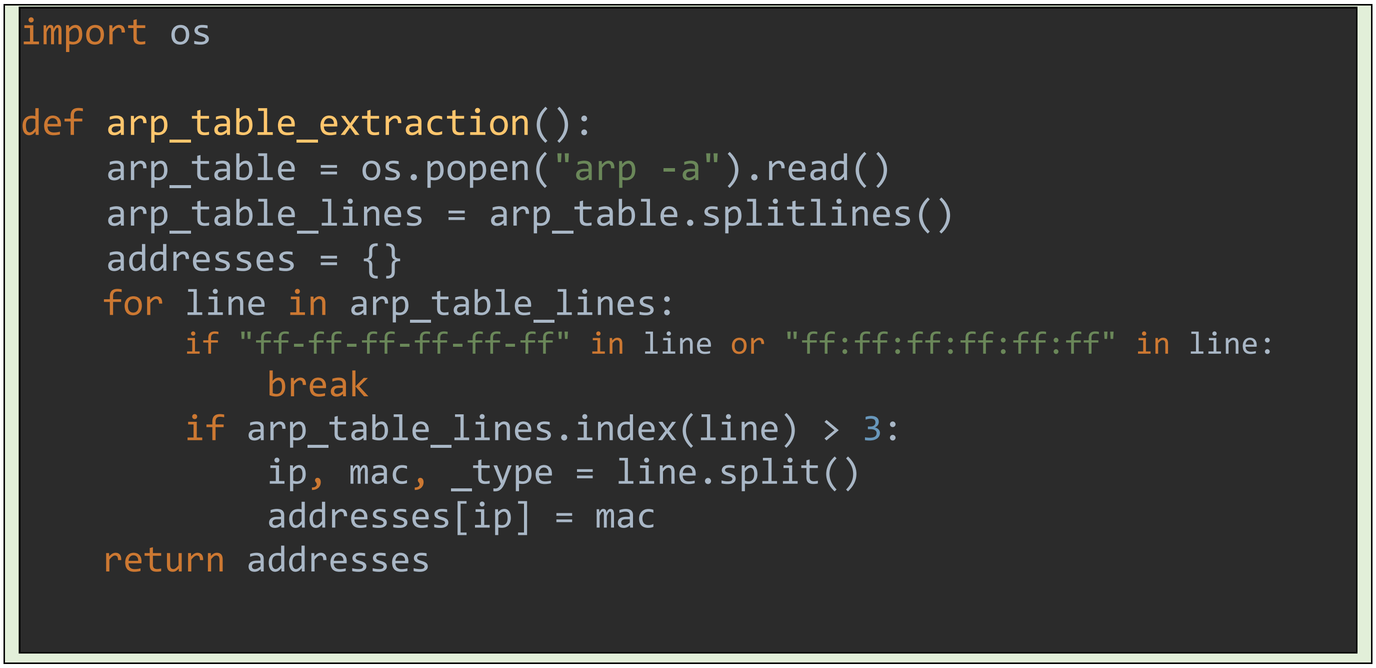
```python
import os

def arp_table_extraction():
```

4 Create three variables: one to store the ARP table data, another to store a list of the separated lines, and the third to store the final filtered data.



```python
import os

def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
```

5 Iterate over the lines and save the IP addresses and corresponding MAC addresses after data filtration. Only IP and MAC addresses should be saved in the third variable. Filter the rest of the data, such as the interface’s IP address or broadcast data.



```python
import os

def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
    for line in arp_table_lines:
        if "ff-ff-ff-ff-ff-ff" in line or "ff:ff:ff:ff:ff:ff" in line:
            break
        if arp_table_lines.index(line) > 3:
            ip, mac, _type = line.split()
            addresses[ip] = mac
    return addresses
```

Project Task 2: Identifying MAC Address Duplication

The second part of the program must take the extracted data and identify if an ARP Spoofing attack is underway. This is done by locating MAC address duplications in the ARP table entries.

1 Plan a function that will identify MAC address duplication.

The function should accept the dictionary generated by the previous steps and iterate over it.

Every iterated MAC address should be saved to a temporary list. The list will then be used to find the MAC address duplications.

2 Define a function to identify duplication in MAC addresses. The function should accept a parameter.

```
def identify_duplication(addresses):
```

3 Create a variable to store iterated MAC addresses for later comparison.

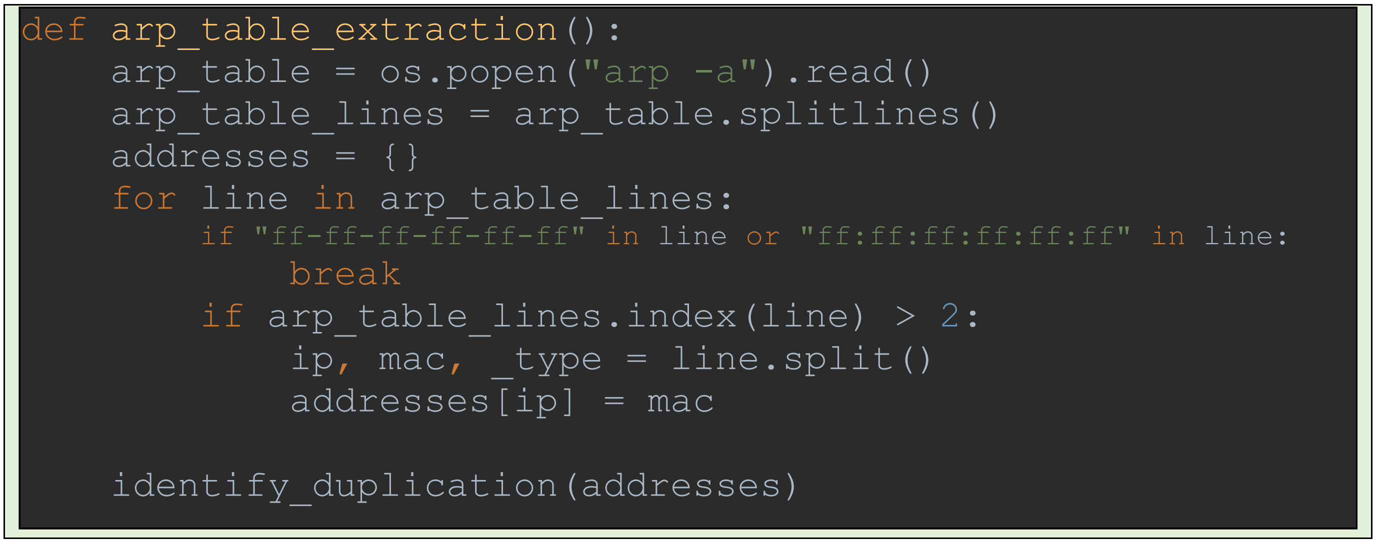
```
def identify_duplication(addresses):
    tmp_mac_lst = []
```

4 Iterate over the recorded MAC addresses and compare them to the saved ones to identify duplications. Print a message that notifies when duplication is identified.

The sleep method is used to perform the calculation in the following code. To use this method, the time module must be imported.

```
def identify_duplication(addresses):
    tmp_mac_lst = []
    print("Scanning...")
    time.sleep(3)
    for mac in addresses.values():
        if mac in tmp_mac_lst:
            print("Arp Spoofed!", "\nThe address is:", mac)
            break
        tmp_mac_lst.append(mac)
```

5 The ARP extracting function passes the filtered data to the current function.



```
def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
    for line in arp_table_lines:
        if "ff-ff-ff-ff-ff-ff" in line or "ff:ff:ff:ff:ff:ff" in line:
            break
        if arp_table_lines.index(line) > 2:
            ip, mac, _type = line.split()
            addresses[ip] = mac

    identify_duplication(addresses)
```

Project Task 3: Logging Events

The third part of the program will handle the logging of the ARP Spoof activity.

1 Plan a function that will log every ARP Spoofing event and save it to a file.

The function should accept a message from the ***identify_duplication*** function.

To act as a legitimate log record, it should also add the event's time to the record, which will require importing the ***datetime*** module.

The function must also create a file and append all data regarding the event.

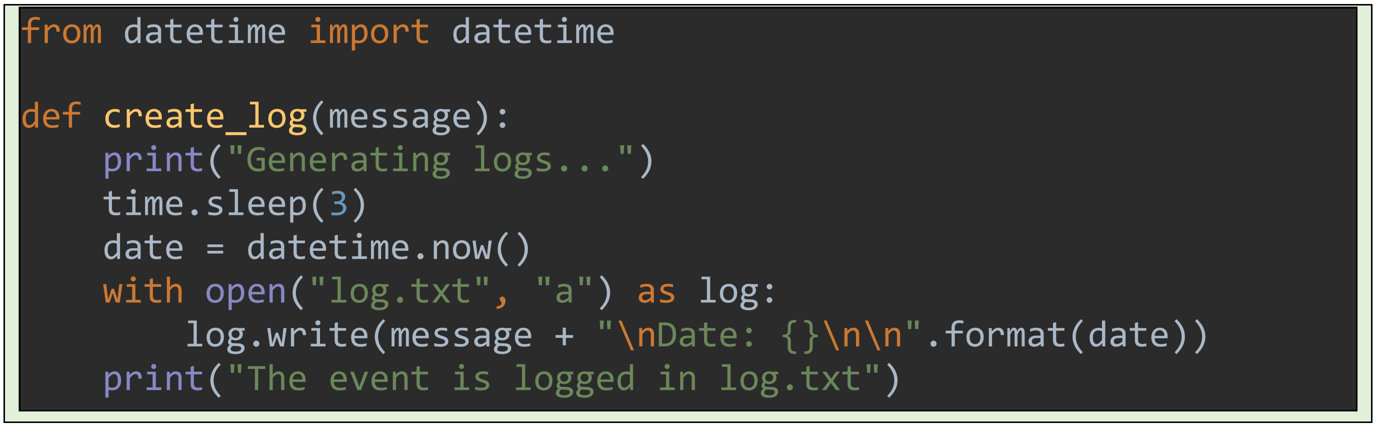
2 Define a function to handle ARP Spoofing event logging. The function should accept data regarding the event.

```
from datetime import datetime

def create_log(message):
```

3 Create a variable to store the date and time of the event.

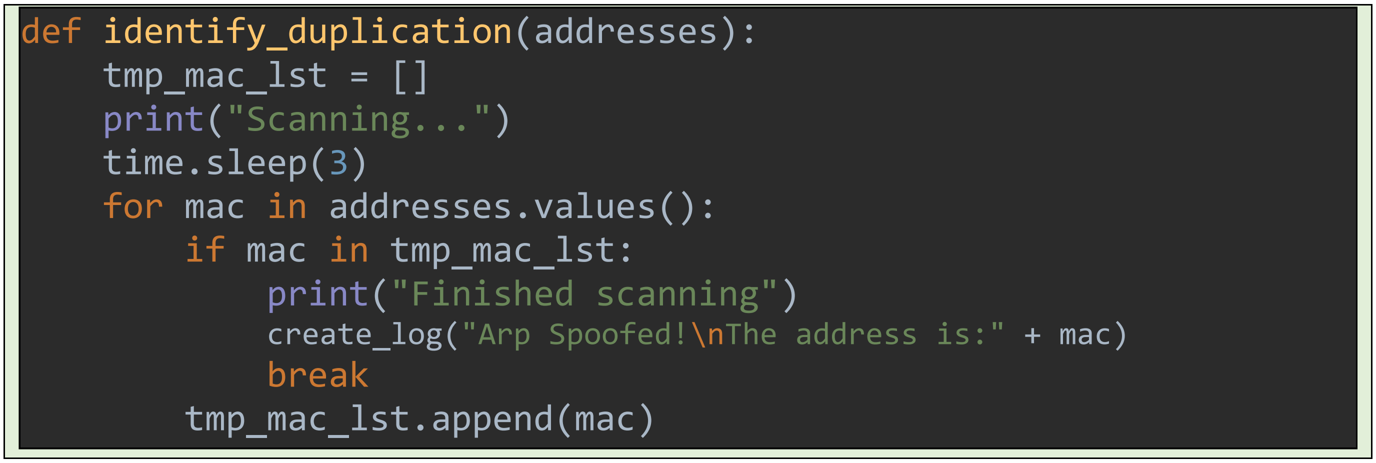
```
from datetime import datetime

def create_log(message):
      date = datetime.now()
```

4 Save the logged data to a file.



```python
from datetime import datetime

def create_log(message):
    print("Generating logs...")
    time.sleep(3)
    date = datetime.now()
    with open("log.txt", "a") as log:
        log.write(message + "\nDate: {}\n\n".format(date))
    print("The event is logged in log.txt")
```

5 The ARP Spoof identification function passes a message to the log creator function.



```python
def identify_duplication(addresses):
    tmp_mac_lst = []
    print("Scanning...")
    time.sleep(3)
    for mac in addresses.values():
        if mac in tmp_mac_lst:
            print("Finished scanning")
            create_log("Arp Spoofed!\nThe address is:" + mac)
            break
        tmp_mac_lst.append(mac)
```

6 Add execution control to ensure the program is executed only if its file is directly executed.

```
import os
import time
from datetime import datetime


def arp_table_extraction():
    arp_table = os.popen("arp -a").read()
    arp_table_lines = arp_table.splitlines()
    addresses = {}
    for line in arp_table_lines:
        if "ff-ff-ff-ff-ff-ff" in line or "ff:ff:ff:ff:ff:ff" in line:
            break
        if arp_table_lines.index(line) > 2:
            ip, mac, _type = line.split()
            addresses[ip] = mac

    identify_duplication(addresses)


def identify_duplication(addresses):
    tmp_mac_lst = []
    print("Scanning...")
    time.sleep(3)
    for mac in addresses.values():
        if mac in tmp_mac_lst:
            print("Finished scanning")
            create_log("Arp Spoofed!\nThe address is:" + mac)
            break
        tmp_mac_lst.append(mac)


def create_log(message):
    print("Generating logs...")
    time.sleep(3)
    date = datetime.now()
    with open("log.txt", "a") as log:
        log.write(message + "\nDate: {}\n\n".format(date))
    print("The event is logged in log.txt")


if __name__ == "__main__":
    arp_table_extraction()
```

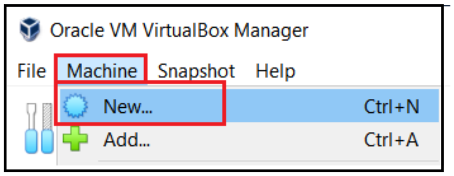
Project Task 4: Testing

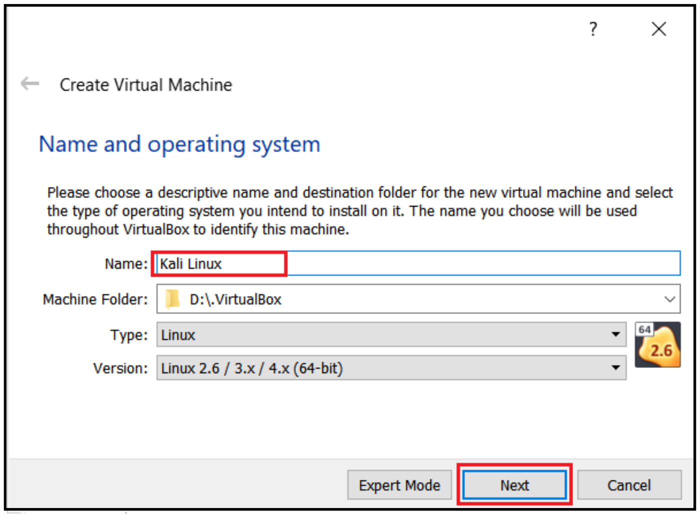
This task is the final step in the project. The program to identify ARP Spoofing activity in the network is complete. The last step is to test its functionality.

To check the script, an ARP Spoof attack must be launched. This can be done from a Kali Linux VM against a Windows host or virtual machine.

The OS should be configured to use the Bridge Adapter network configuration for direct communication with the router.

1 Open VirtualBox, select the **Machine** tab, click ***New…***, and name the VM **Kali Linux**.





Create Virtual Machine

Name and operating system

Please choose a descriptive name and destination folder for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

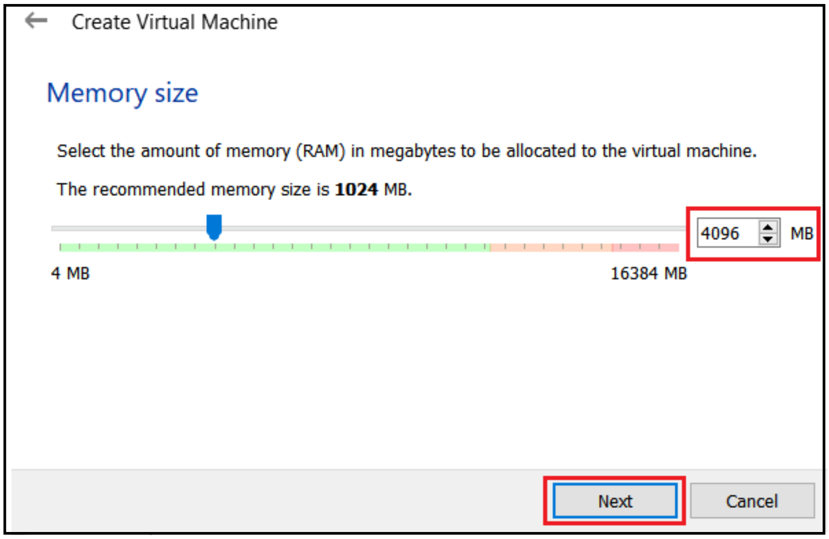
Name: Kali Linux

Machine Folder: D:\.VirtualBox

Type: Linux

Version: Linux 2.6 / 3.x / 4.x (64-bit)

2 Set the memory to **2048 MB** for proper functionality. For enhanced functionality, you can use a higher setting (per the computer's available resources).



Create Virtual Machine

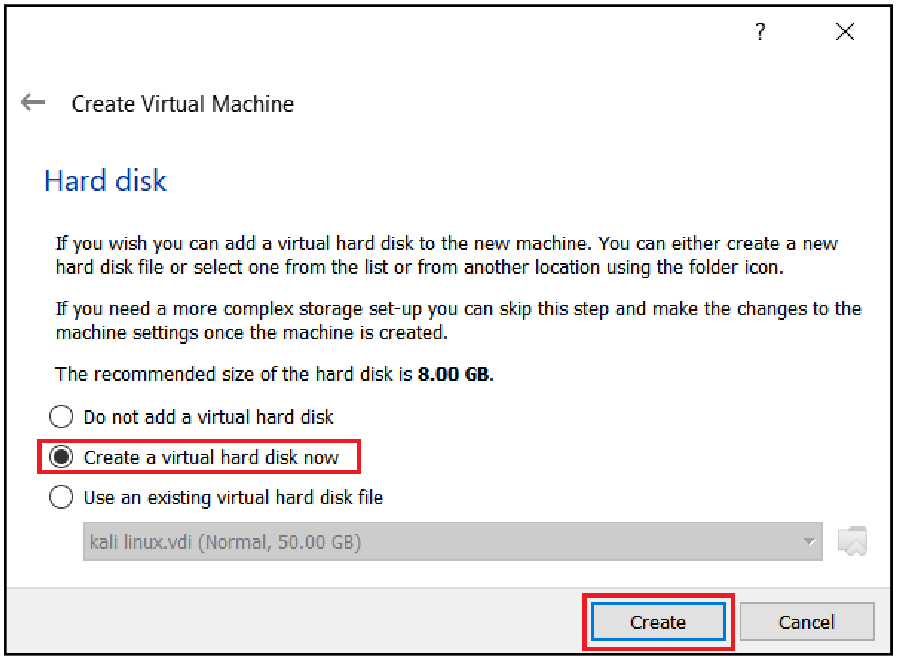
Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

The recommended memory size is **1024** MB.

4096 MB

3 Select ***Create a virtual hard disk now*** in the next window and click **Create**.



Create Virtual Machine

Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

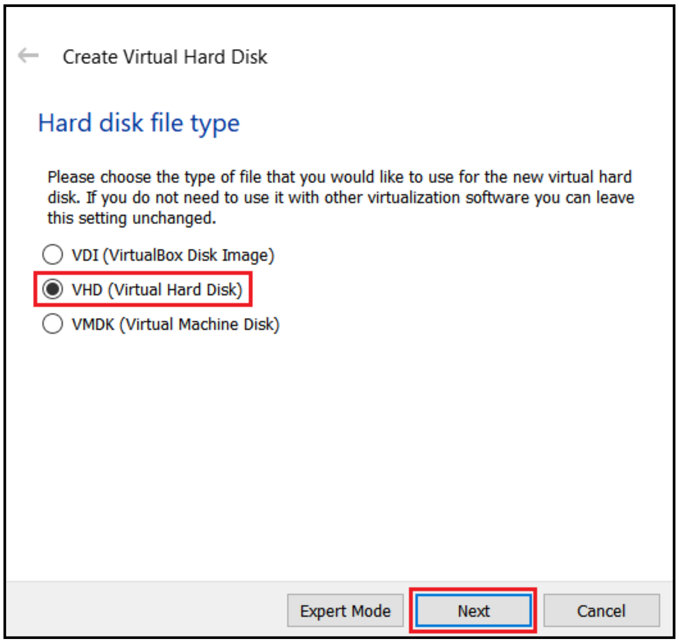
○ Do not add a virtual hard disk

● Create a virtual hard disk now

○ Use an existing virtual hard disk file

kali linux.vdi (Normal, 50.00 GB)

4 Select ***VHD*** for the hard disk file type and click **Next**.



Create Virtual Hard Disk

Hard disk file type

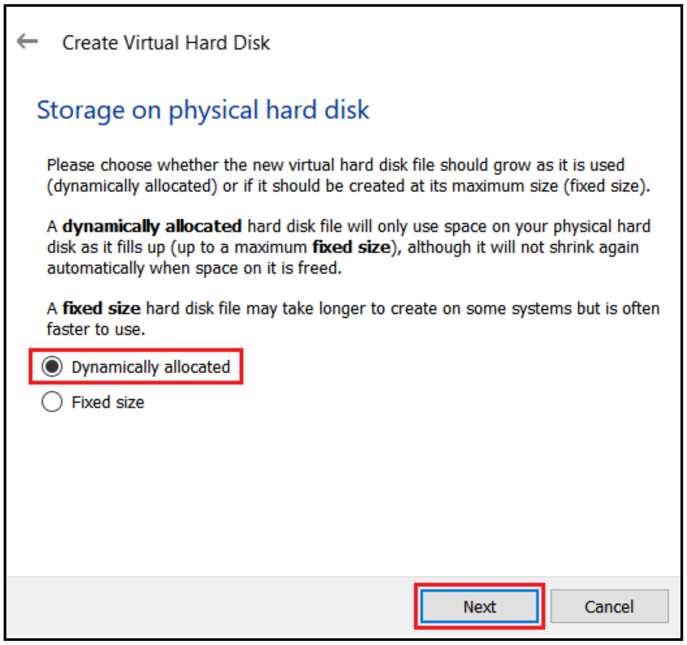
Please choose the type of file that you would like to use for the new virtual hard disk. If you do not need to use it with other virtualization software you can leave this setting unchanged.

○ VDI (VirtualBox Disk Image)

● VHD (Virtual Hard Disk)

○ VMDK (Virtual Machine Disk)

5 Select ***Dynamically allocated*** and click **Next**.



Create Virtual Hard Disk

Storage on physical hard disk

Please choose whether the new virtual hard disk file should grow as it is used (dynamically allocated) or if it should be created at its maximum size (fixed size).

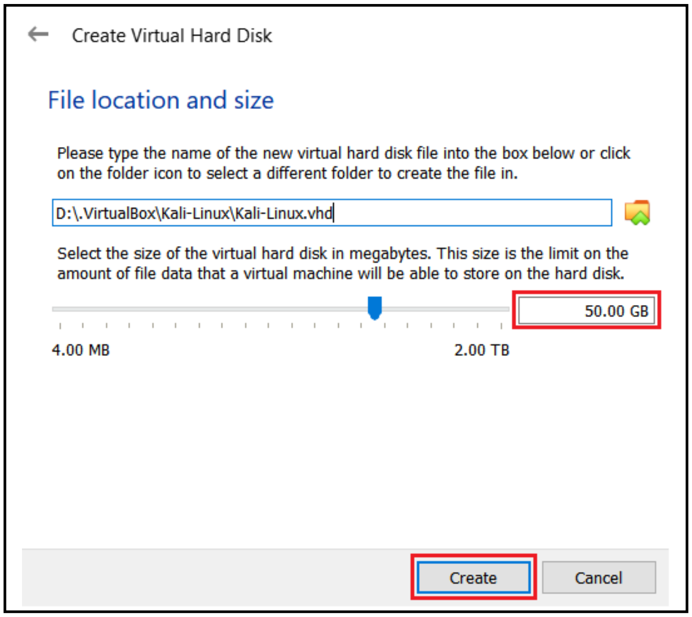
A **dynamically allocated** hard disk file will only use space on your physical hard disk as it fills up (up to a maximum **fixed size**), although it will not shrink again automatically when space on it is freed.

A **fixed size** hard disk file may take longer to create on some systems but is often faster to use.

● Dynamically allocated

○ Fixed size

6 Select the file location and set the storage size to **50 GB**. If your computer does not have enough storage, you can set it to **20 GB**.



Create Virtual Hard Disk

File location and size

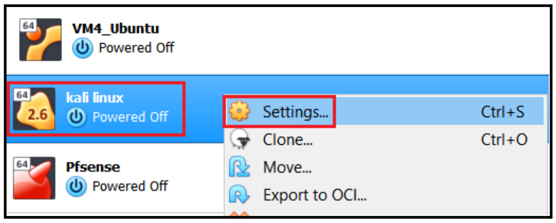
Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

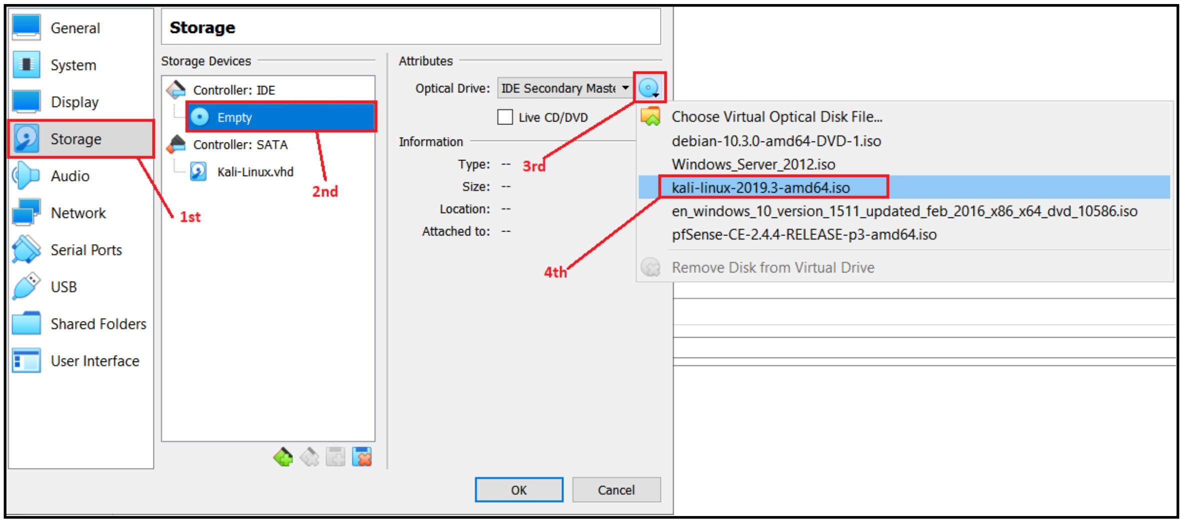
D:\.VirtualBox\Kali-Linux\Kali-Linux.vhd

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

50.00 GB

7 Right-click the Kali VM, select ***Settings…***, and insert the **Kali Linux ISO** file into the drive.





Storage

Optical Drive: IDE Secondary Maste

□ Live CD/DVD

Information

Type: --

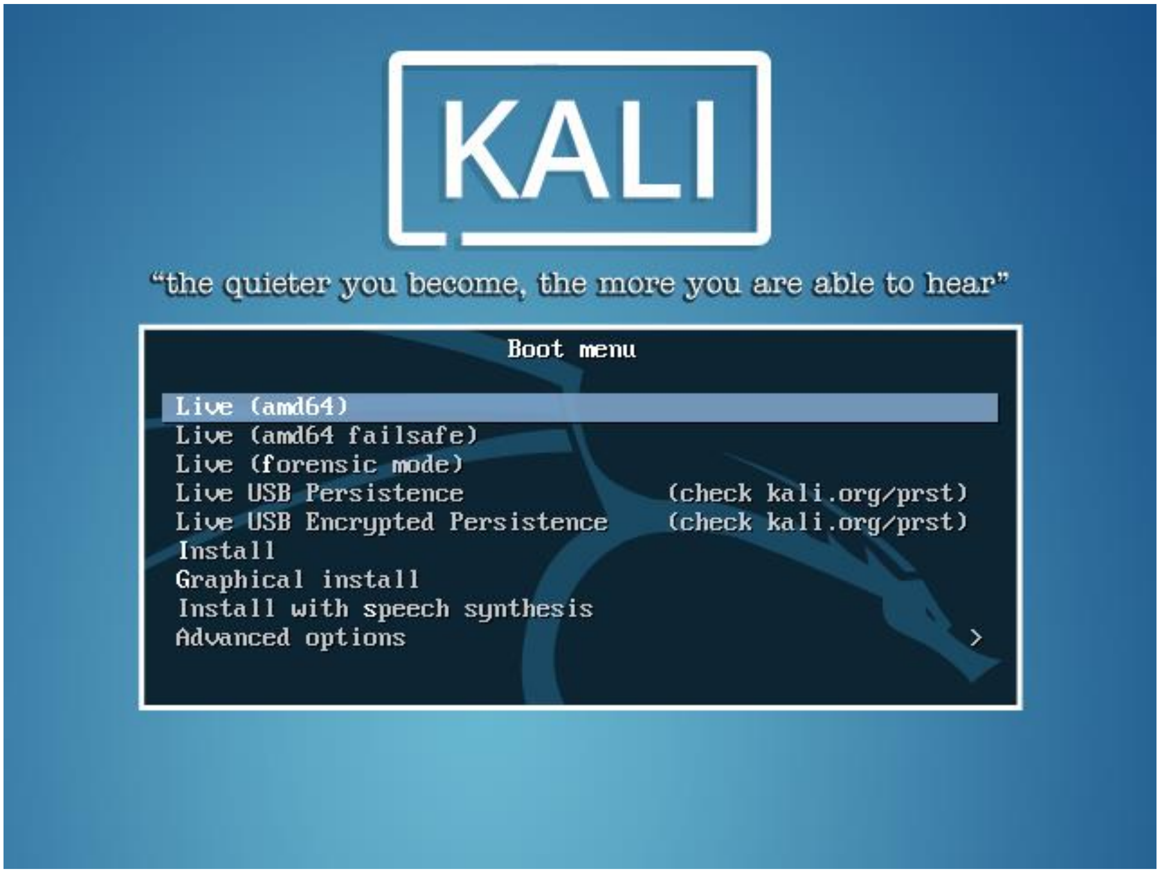
Size: --

Location: --

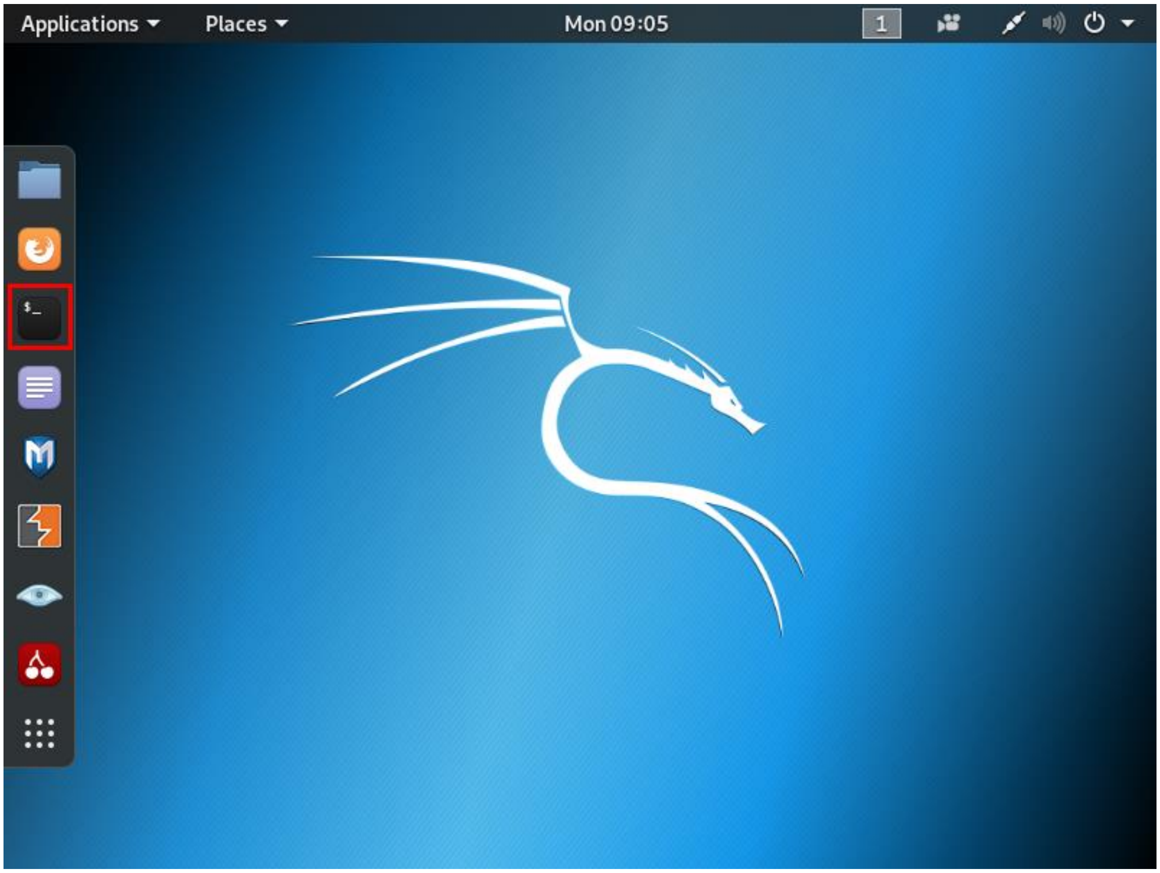
Attached to: --

General
System
Display
Storage
Audio
Network
Serial Ports
USB
Shared Folders
User Interface
Storage Devices
Controller: IDE
Empty
Controller: SATA
Kali-Linux.vhd
1st
2nd
Attributes
3rd
4th
Choose Virtual Optical Disk File...
debian-10.3.0-amd64-DVD-1.iso
Windows_Server_2012.iso
kali-linux-2019.3-amd64.iso
en_windows_10_version_1511_updated_feb_2016_x86_x64_dvd_10586.iso
pfSense-CE-2.4.4-RELEASE-p3-amd64.iso
Remove Disk from Virtual Drive
OK
Cancel

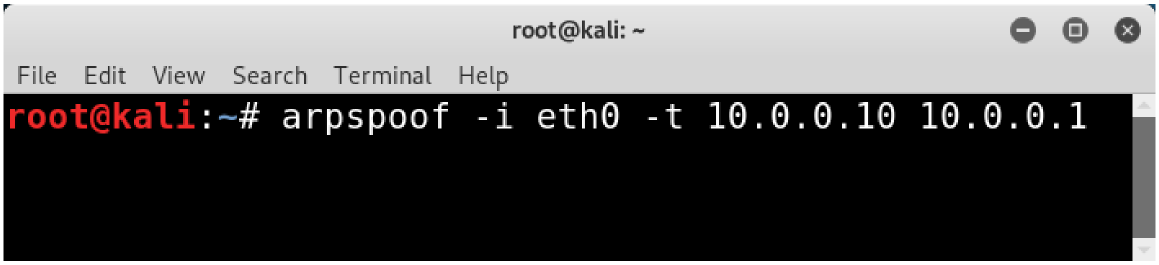
8 Run the virtual machine and select the live version.


KALI
“the quieter you become, the more you are able to hear”
Boot menu
Live (amd64)
Live (amd64 failsafe)
Live (forensic mode)
Live USB Persistence (check kali.org/prst)
Live USB Encrypted Persistence (check kali.org/prst)
Install
Graphical install
Install with speech synthesis
Advanced options >

9 Open the terminal by clicking on it in the options on the left.

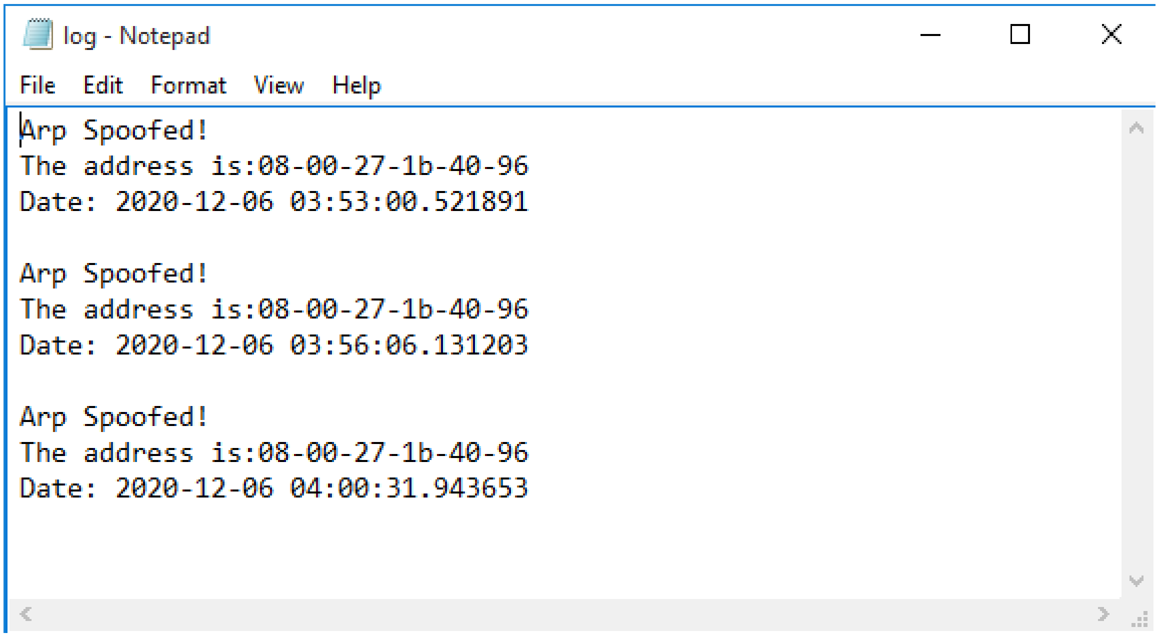


10 Run the ***arpspoof -i eth0 -t <target IP> <spoofed IP>*** command.



```
root@kali:~# arpspoof -i eth0 -t 10.0.0.10 10.0.0.1
```

11 Execute the script and note that a log was created.



```
Arp Spoofed!
The address is:08-00-27-1b-40-96
Date: 2020-12-06 03:53:00.521891

Arp Spoofed!
The address is:08-00-27-1b-40-96
Date: 2020-12-06 03:56:06.131203

Arp Spoofed!
The address is:08-00-27-1b-40-96
Date: 2020-12-06 04:00:31.943653
```